

Software project management

and

Maven

Problem area

- Large software projects usually contain tens or even hundreds of projects/modules
- Will become messy and incomprehensible if the projects don't adhere to some common principles
- Will be time-consuming to build all projects manually

The preferred solution

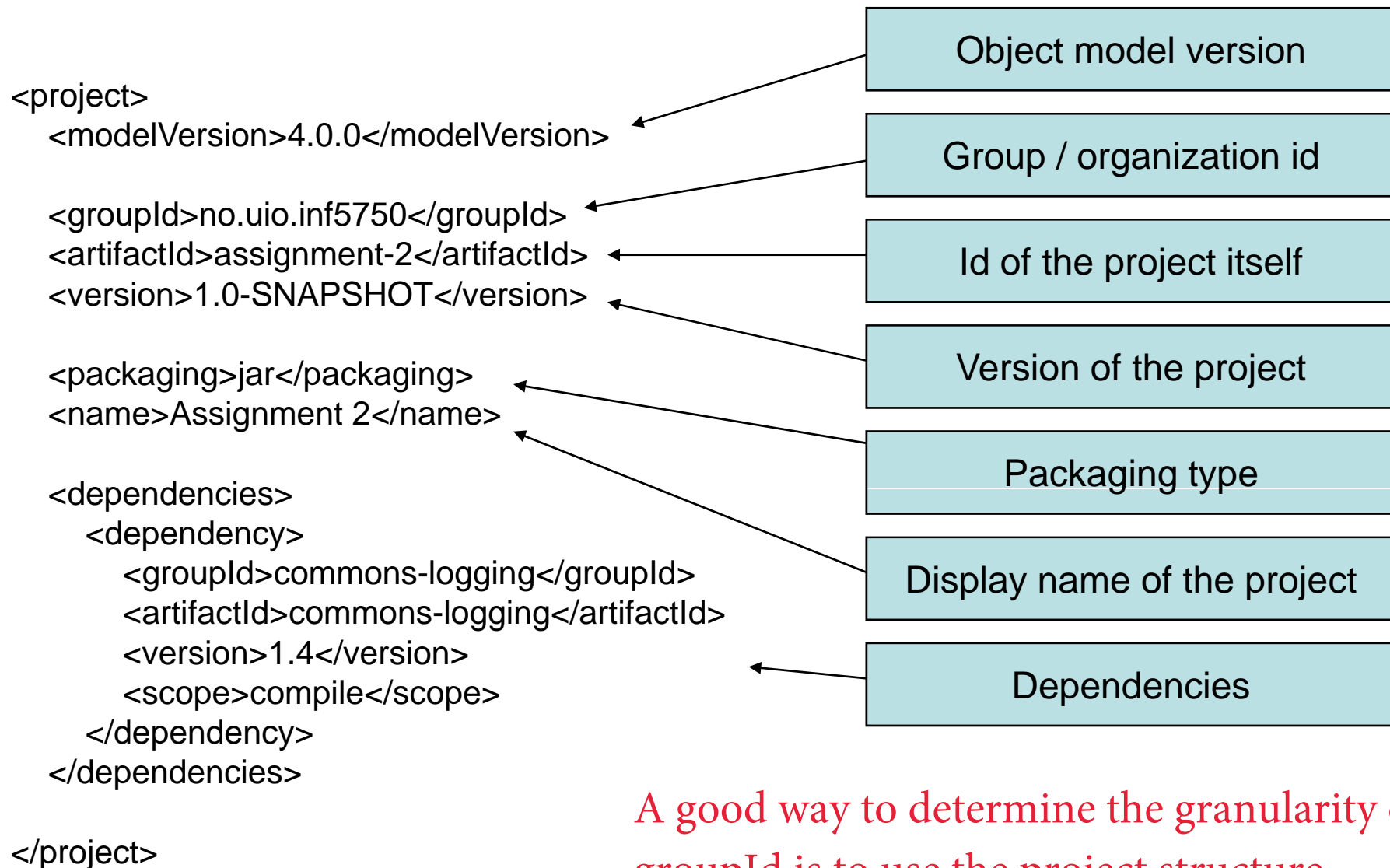
- Use a project management tool (like Maven)
- Maven helps you with various aspects:
 1. Build process
 2. Project structure
 3. Dependency management
 4. Access to information and documentation

1. Build process

- The Project Object Model (POM) – an XML file – is the heart of a Maven 2 project
- Contains project information and configuration details used to build the project
 - Project dependencies
 - Commands (goals) that can be executed
 - Plugins
 - Metadata
- The POM extends the Super POM
 - Only 4 lines are required

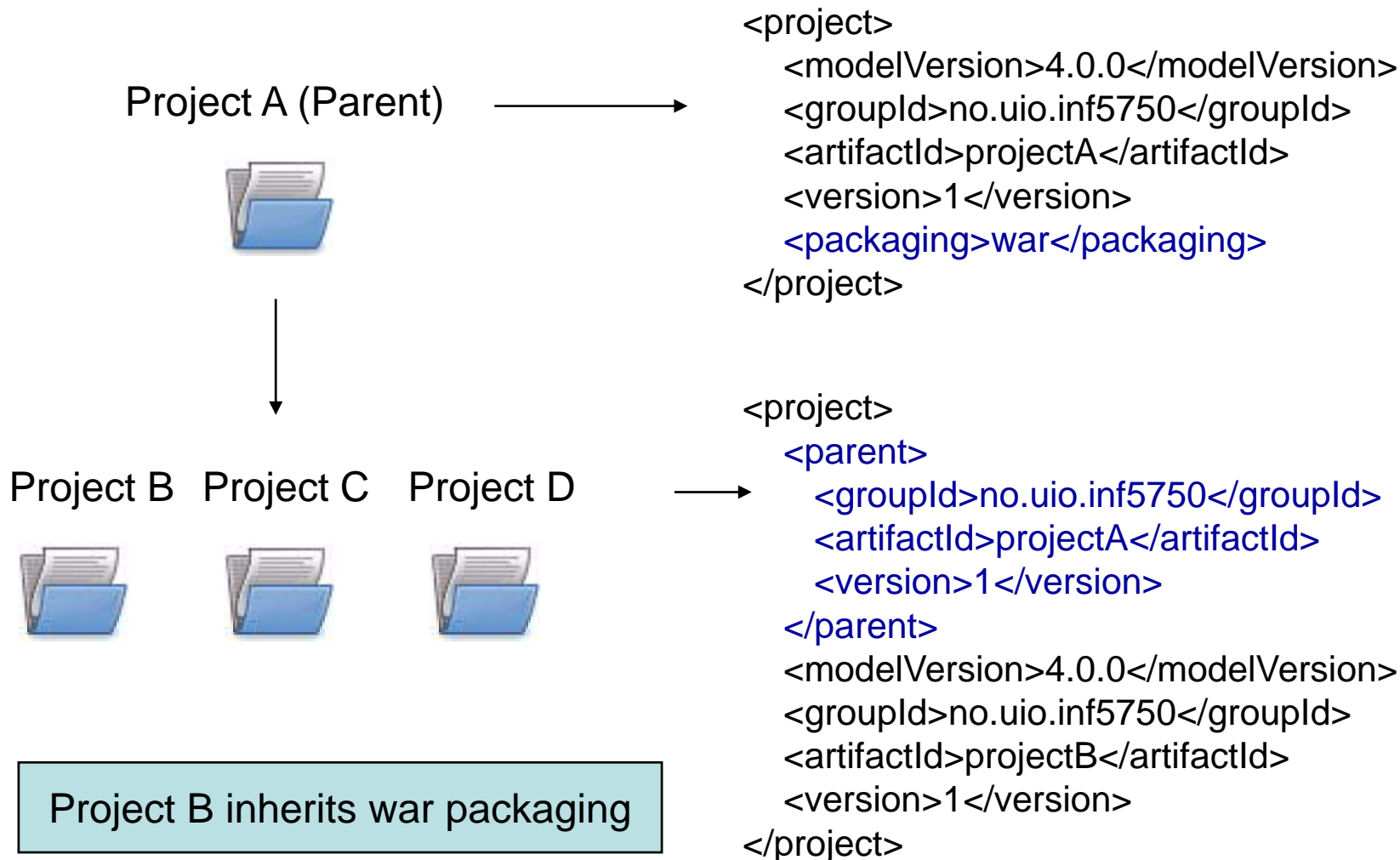
```
<modelVersion>4.0.0</modelVersion>  
<groupId>org.codehaus.mojo</groupId>  
<artifactId>my-project</artifactId>  
<version>1.0</version>
```

1. POM - Simple example

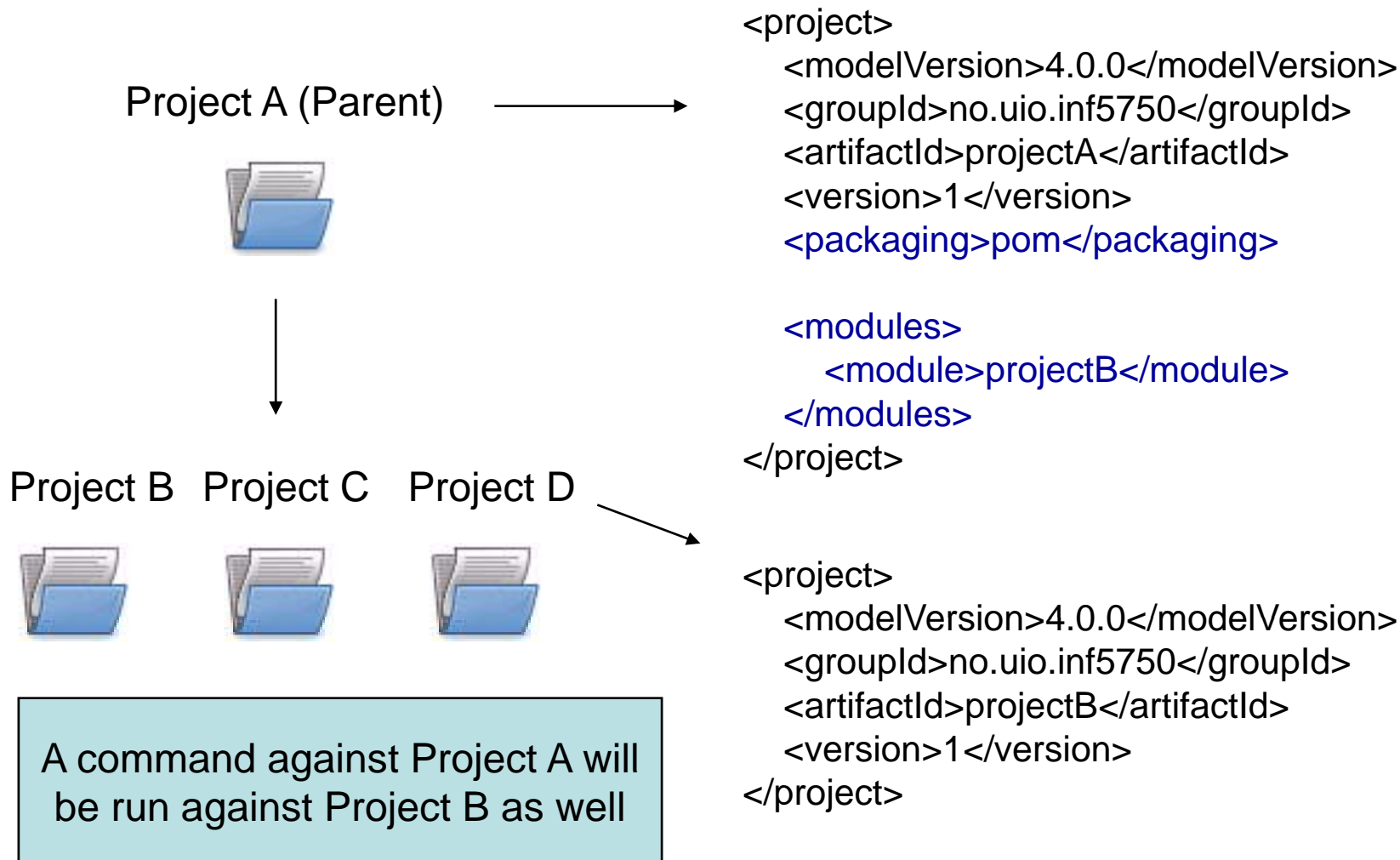


A good way to determine the granularity of the `groupId` is to use the project structure

1. POM – Project inheritance



1. POM – Project aggregation



1. Build Lifecycle and Phases

- The *build lifecycle* is the process of building and distributing an artifact
- A *phase* is a step in the build lifecycle
- Most important default phases:
 - Validate
 - Compile
 - Test
 - Package
 - Install
 - Deploy
- Some common phases *not* default:
 - Clean
 - Site
- For each step, all previous steps are executed

2. Standard directory layout

- Advantages:
 - A developer familiar with Maven will quickly get familiar with a new project
 - No time wasted on re-inventing directory structures and conventions

src/main/java

Java source files goes here

src/main/resources

Other resources your application needs

src/main/filters

Resource filters (properties files)

src/main/config

Configuration files

src/main/webapp

Web application directory for a WAR project

src/test/java

Test sources like unit tests (not deployed)

src/test/resources

Test resources (not deployed)

src/test/filters

Test resource filter files (not deployed)

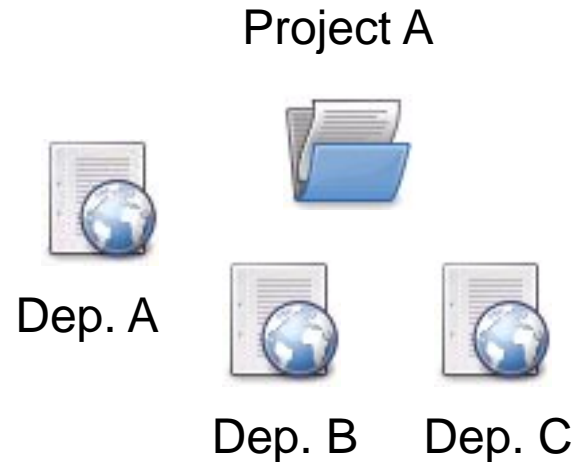
src/site

Files used to generate the Maven project website

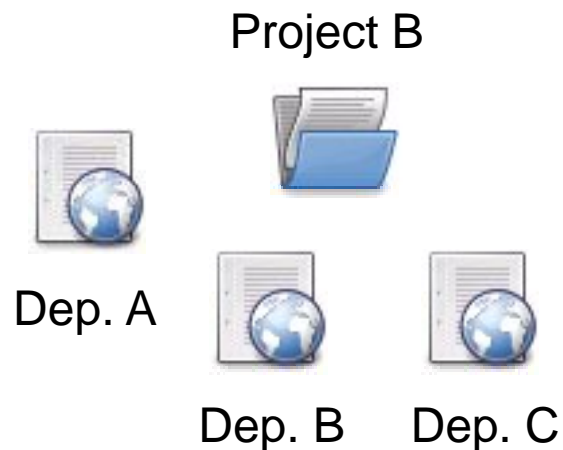
3. Dependency management

- Dependency: a third-party or project-local software library (JAR or WAR file)
- Dependency management is a challenge in multi-module projects

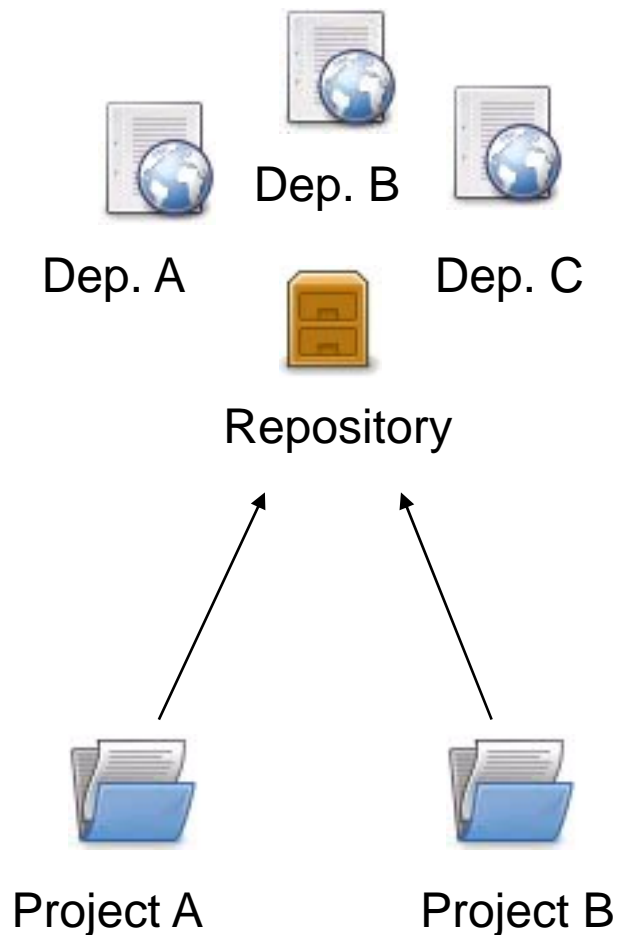
3. Dependency management



- The poor approach: Replicate all dependencies for every project (put in /lib folder within the project)
 - Dependencies are replicated and use more storage
 - Checking out a project will be slow
 - Difficult to keep track of versions



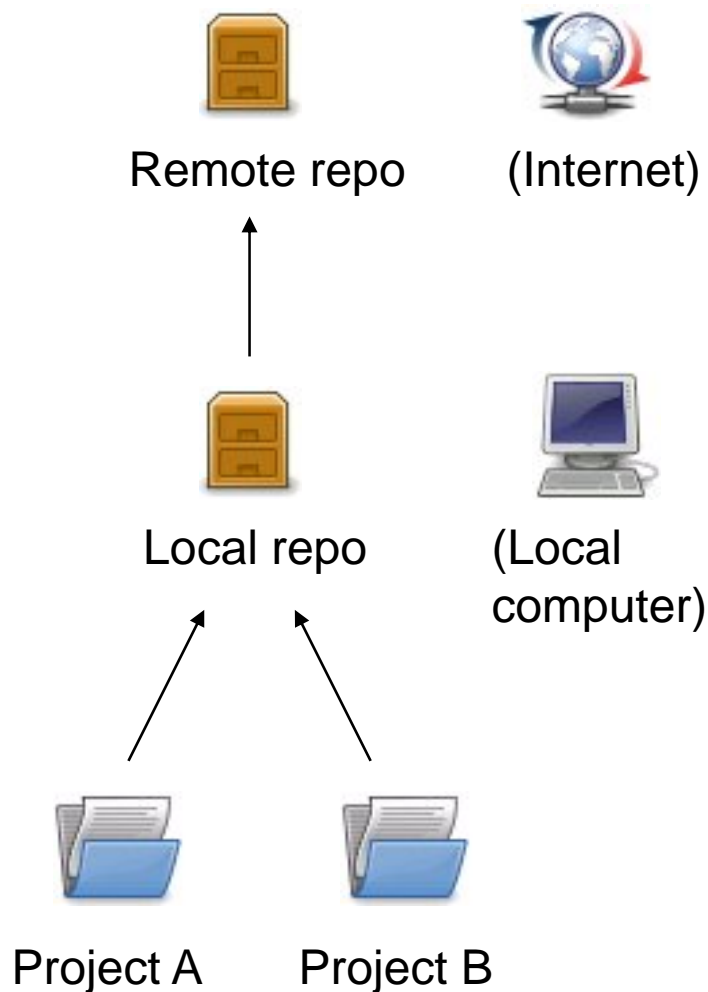
3. Dependency management



- The preferred solution: Use a *repository*
- Repository: A shared location for dependencies which all projects can access
 - Only one copy exists
 - Stored outside the project
- Dependencies are defined in the POM

```
<dependencies>
  <dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.3</version>
  </dependency>
</dependencies>
```

3. Repositories



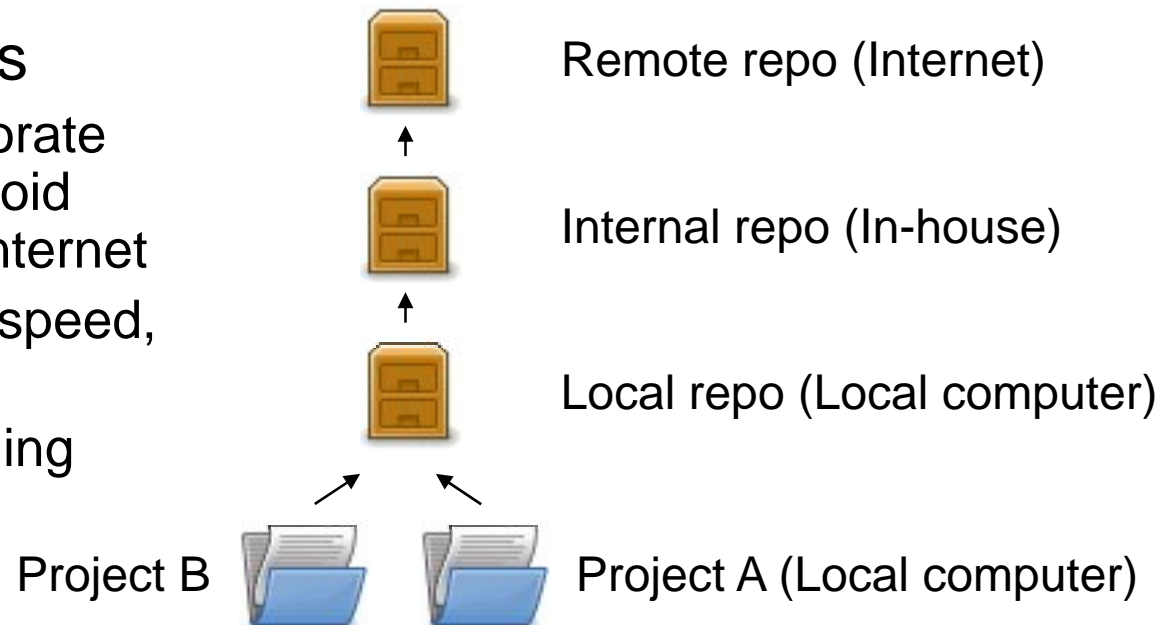
- Remote repository:
 - Provides software artifacts (dependencies) for download
 - E.g. [repo1.maven.org](https://repo1.maven.org/maven2/) houses Maven's central repository
- Local repository:
 - Copy on local computer which is a cache of the remote downloads
 - May contain project-local build artifacts as well
 - Located in `USER_HOME/.m2/repository`
 - Same structure as remote repos

3. Repositories

- Downloading from a remote repository
 - Central repo is default
 - Can be overridden

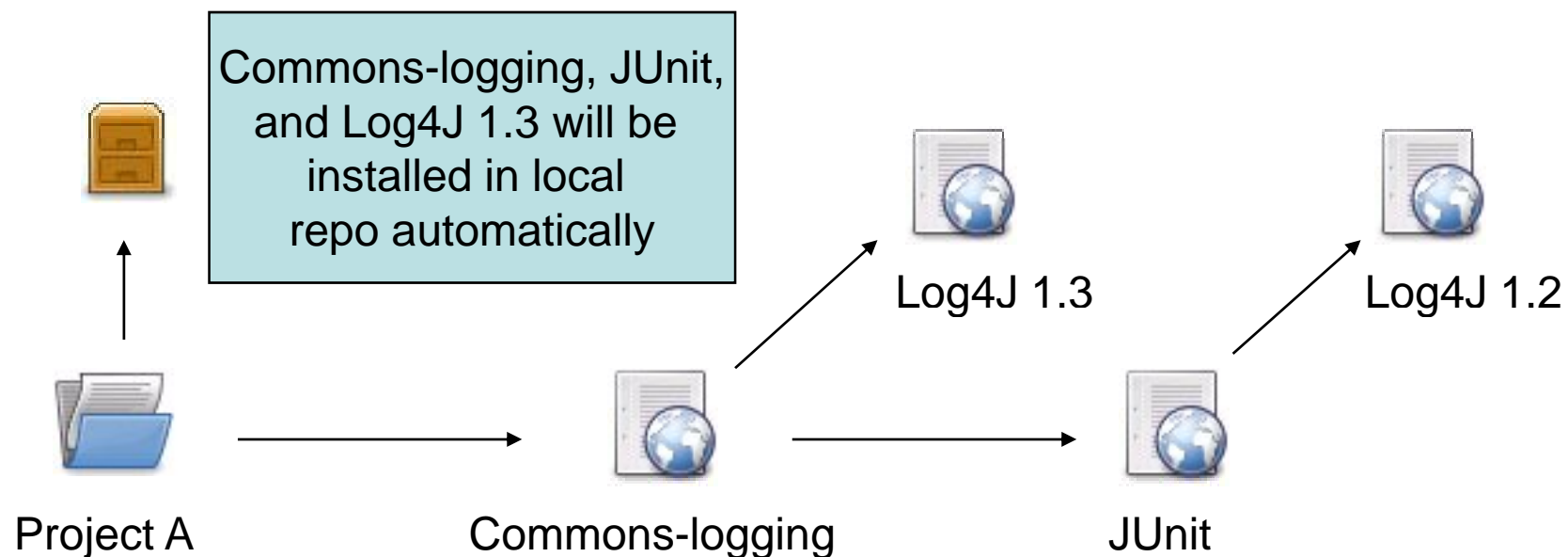
```
<repositories>  
  <repository>  
    <id>my-repo-</id>  
    <url>http://my-server/repo</url>  
  </repository>  
</repositories>
```

- Internal repositories
 - Often used in corporate environments to avoid connection to the internet
 - Improves security, speed, and bandwidth
 - Suitable for publishing private artifacts



3. Transitive dependencies

- Maven reads the POM files of your dependencies and automatically includes their required libraries
- No limit on the number of levels
- *Dependency mediation* – nearest definition



3. Dependency scope

- Affects the classpath used for various build tasks
- Can be defined for all dependencies, *compile* default
- 5 dependency scopes available:
 - Compile: Available in all classpaths (default)
 - Provided: The JDK or the container provides it
 - Runtime: Only required for execution, not for compilation
 - Test: Only required for testing, not for normal use (not deployed)
 - System: You provide it locally, not looked up in a repo

```
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.4</version>
  <scope>compile</scope>
</dependency>
```


3. Dependency management

- Mechanism for centralizing dependency information
- Favourable for projects that inherits a common parent
- Useful for controlling versions of transitive dependencies

Parent POM

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.0</version>
      <scope>test</scope>
      <type>jar</type>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Child POMs

```
...
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
</dependency>
...
```

Child POM dependency
inherits information
from parent POM

Transitive occurrences
of JUnit guaranteed to
be of version 4.0

4. Project information

- Powerful feature in Maven: Create a project site automatically
- Info retrieved from the POM, source code
- Provides information regarding
 - Dependencies
 - Issue tracking
 - Licensing
 - Development team
- Provides various reports
 - Test coverage
 - Internationalisation
 - JavaDocs
 - Potential code problems

Useful commands

- `$ mvn package` Compile and create JARs/WARs
- `$ mvn install` Package + copy to local repo
- `$ mvn clean` Delete target directory
- `$ mvn test` Run unit tests

- `$ mvn eclipse:eclipse` Create Eclipse project files
- `$ mvn idea:idea` Create IDEA project files

- `$ mvn jetty:run-war` Run a WAR file in Jetty
- `$ mvn site` Generates project site

Summary

- We've learned that Maven facilitates:
 - Uniform building of projects through the POM
 - Consistent project structure
 - Management of dependencies through repositories to avoid replication and ease re-use and versioning
 - Standardized and professional-quality project information

Resources

- "Better builds with Maven"
 - Free PDF book online
 - <http://www.devzuz.com/>
- Maven homepage
 - Documentation and guides
 - <http://maven.apache.org>